A

The Screen Behind the Mirror

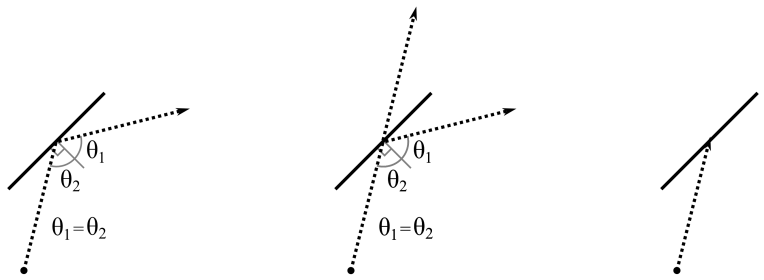
Introduction:

Dr. Evil has contracted your valuable services to build for him the world's most powerful "laser". Of course before you spend one billion dollars building the thing, you want to run some simulations first to make sure everything will work as designed. For this phase of the project, you will be simulating part of the aiming system which uses mirrors and other optics to change the direction of the laser beam.

The simulation consists of a flat square table with mirrors, beam splitters, and beam detectors arranged on the tabletop, and with each object represented by a one dimensional line segment. The list below describes each of the object types in detail:

- *mirror* : A mirror object will reflect any laser beam striking its surface. The reflected beam leaves at the same angle of incidence as the incoming beam. Note that **both sides of a mirror object are reflective**.
- *detector* : A detector is an opaque object which absorbs any laser beam striking it. The simulation must also keep track of which detectors are struck by a laser for program output purposes. Note that a laser beam strike on either side of a detector counts as a "hit".
- *splitter* : When a laser beam strikes a splitter, it divides into two beams. One of the new beams will reflect from the splitter surface (as with a mirror), and the other beam will pass through the splitter without changing direction. A splitter will function the same way regardless which side of it is struck by a laser beam.

See the figures below for examples of a laser beam's interaction with each of the possible object types:



Mirror Object *Splitter Object* *Detector Object*

For each simulation, a single laser beam enters the tabletop area. The program must compute the path taken by the laser beam (including secondary beams due to splitters), and it must determine which detectors are struck by a laser beam.

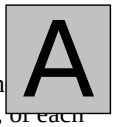
You can make the following assumptions in the program:

1. The tabletop surface is a 100 by 100 square, and unless otherwise specified all coordinates in the program's input are given as integers within the tabletop area (i.e. between 0 and 100 inclusive).
2. There will be no overlaps between the line segment objects.
3. The laser which enters the tabletop area always starts from the edge of the table.
4. The simulation of each data set ends when all laser beams have either exited the table top area or have terminated at a detector.
5. For each data set there will be no more than 100 total reflections among all laser beams in that data set.
6. A laser beam will never intersect any object on a vertex and will never be collinear with an object's line segment.
7. Each data set will contain at least one detector object.

Input:

Input to this problem will begin with a line containing a single integer N ($1 \le N \le 100$) indicating the number of data sets. Each data set consists of the following components:

- A single line with four numbers "x,y i,j" where x,y is a point along the table edge at which the laser beam enters, and i,j is a vector with integer components ($-1024 \le i,j \le 1024$) specifying the direction of the incoming laser beam, where i corresponds to the x-axis direction and j corresponds to the y-axis direction.
- A line with a single integer P ($1 \le P \le 100$) giving the total number of objects in this data set.
- A series of P lines, each representing one object, with the first line describing object 1, the second line describing object 2, and so on. Each line begins with a single letter specifying the object type where a "`M`" indicates a mirror object, "`S`" a splitter, and "`D`" a detector. This is followed by two coordinate pairs of the form "x,y", specifying the two end points of the object's line segment.



Output:

For each data set in the input, output the heading "`DATA SET #`*k*" where *k* is 1 for the first data set, 2 for the second, etc. If in this data set non-detector objects are struck by any laser beams, output the message "`NO BEAMS DETECTED`". Otherwise, output the object number, one per line, of each detector struck by a laser beam. The list of detectors should be sorted by their object numbers and output in ascending order. If a detector is struck by more than one laser beam, it should only be listed once in the output.

Sample Input:

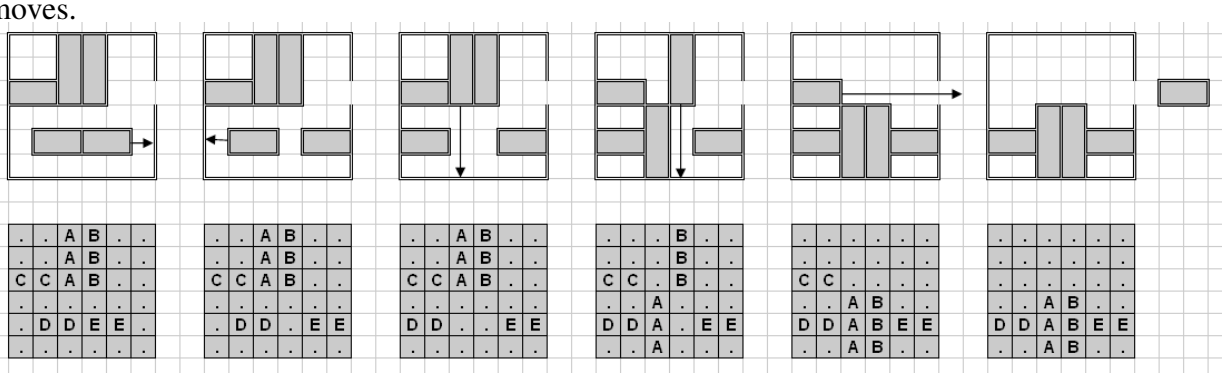
```
1
50,100 0,-1
6
D 0,40 20,20
M 40,20 60,40
D 80,20 100,40
D 0,70 20,90
S 40,90 60,70
D 80,90 100,70
```

Sample Output:

```
DATA SET #1
1
6
```

Problem B: Block Game

Bud bought this new board game. He is hooked. He has been playing it over and over again such that he thinks can finish the game with the minimum number of moves, but he is uncertain. He wants you to help him check whether the moves he has listed are indeed the minimum number of moves.



You are given a 6x6 board, and a set of 2x1 or 3x1 (vertical) or 1x2 or 1x3 (horizontal) pieces. You can slide the horizontal pieces horizontally only, and the vertical pieces vertically only. You are only allowed to slide a piece if there is no other piece, nor a wall, obstructing its path.

There will be one special 1x2 horizontal piece. There will also be a gap in the wall, on the right side, on the same row as the special piece, that *only* the special piece can fit through. The goal of the game is to get that one special horizontal piece out of the gap on the right side.

A "move" in this game is when you take a piece and slide it any number of squares (i.e. if you slide a piece horizontally one square, that is one move, and sliding it 2 squares at once is also considered one move).

Input

Input will consist of multiple datasets. Each data set will begin with a line with a single capital letter, indicating the special piece which must move off of the board.

The next 6 lines will consist of 6 characters each. These characters will either be a '.' (period), indicating an empty square, or a capital letter, indicating part of a piece.

The letters are guaranteed to form pieces that are 1x2, 1x3, 2x1 or 3x1, and no letter will be used to represent more than one piece on any given board. The letter indicating the special piece is guaranteed to correspond to a 1x2 piece somewhere on the board.

The end of data is indicated by a single '*' (asterisk) on its own line.

Output

For each data set, print a single integer, indicating the smallest number of moves necessary to remove the given piece, or -1 if it isn't possible. Print each integer on its own line. There should be no blank lines between outputs.

Example

Given the input

```
C
..AB..
..AB..
CCAB..
......
.DDEE.
......
A
......
......
......
......
AA....
......
Z
.ZZ..X
.....X
.....X
.....Y
.....Y
.....Y
*
```

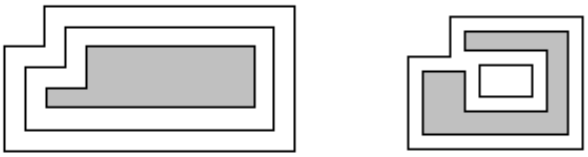
the output would be

```
5
1
-1
```

C

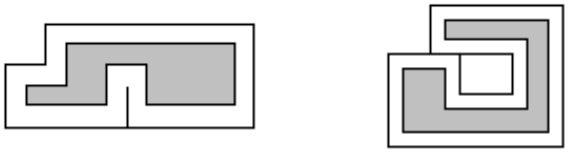
Problem A: Bordering on Madness

Bob Roberts owns a design business which creates custom artwork for various corporations. One technique that his company likes to use is to take a simple rectilinear figure (a figure where all sides meet at 90 or 270 degrees and which contains no holes) and draw one or more rectilinear borders around them. Each of these borders is drawn so that it is a set distance d away from the previously drawn border (or the original figure if it is the first border) and then the new area outlined by each border is painted a unique color. Some examples are shown below (without the coloring of the borders).



The example on the left shows a simple rectilinear figure (grey) with two borders drawn around it. The one on the right is a more complicated figure; note that the border may become disconnected.

These pieces of art can get quite large, so Bob would like a program which can draw prototypes of the finished pieces in order to judge how aesthetically pleasing they are (and how much money they will cost to build). To simplify things, Bob never starts with a figure that results in a border where 2 horizontal (or vertical) sections intersect, even at a point. This disallows such cases as those shown below:



Input

Input will consist of multiple test cases. The first line of the input file will contain a single integer indicating the number of test cases. Each test case will consist of two or more lines. The first will contain three positive integers n, m and d indicating the number of sides of the rectlinear figure, the number of borders to draw, and the distance between each border, where $n \leq 100$ and $m \leq 20$. The remaining lines will contain the n vertices of the figure, each represented by two positive integers indicating the x and y coordinates. The vertices will be listed in clockwise order starting with the vertex with the largest y value and (among those vertices) the smallest x value.

Output

For each test case, output three lines: the first will list the case number (as shown in the examples), the second will contain m integers indicating the length of each border, and the third will contain m integers indicating the additional area contributed to the artwork by each border. Both of these sets of numbers should be listed in order, starting from the border nearest the original figure. Lines two and three should be indented two spaces and labeled as shown in the examples. Separate test cases with a blank line.

Sample Input

```
2
6 2 10
20 30 100 30 100 0 0 0 0 10 20 10
10 1 7
20 50 70 50 70 0 0 0 0 30
20 30 20 10 60 10 60 40 20 40
```

Sample Output

```
Case 1:
  Perimeters: 340 420
  Areas: 3000 3800

Case 2:
  Perimeters: 380
  Areas: 2660
```

D

Another Brick in the Wall

Introduction

After years as a brick-layer, you've been called upon to analyze the structural integrity of various brick walls built by the Tetrad Corporation. Instead of using regular-sized bricks, the Tetrad Corporation seems overly fond of bricks made out of strange shapes. The structural integrity of a wall can be approximated by the fewest number of bricks that could be removed to create a gap from the top to the bottom. Can you determine that number for various odd walls created by Tetrad?

Input

Input to this problem will begin with a line containing a single integer X ($1 \leq X \leq 100$) indicating the number of data sets. Each data set consists of two components:

- A single line, "M N" ($1 \leq M,N \leq 20$) where M and N indicate the height and width (in units), respectively, of a brick wall;
- A series of M lines, each N alphabetic characters in length. Each character will indicate to which brick that unit of the wall belongs to. Note that bricks will be contiguous; each unit of a brick will be adjacent (diagonals do not count as adjacent) to another unit of that brick. Multiple bricks may use the same characters for their representation, but any bricks that use identical characters will not be adjacent to each other. All letters will be uppercase.

Output

For each data set, output the fewest number of bricks to remove to create a gap that leads from some point at the top of the wall, to some point at the bottom of the wall. Assume that bricks are in fixed locations and do not "fall" if bricks are removed from beneath them. A gap consists of contiguous units of removed bricks; each unit of a gap must be adjacent (diagonals do not count) to another unit of the gap.

Sample Input

```
3
5 7
AABBCCD
EFFGGHH
IIJJKKL
MNNOOPP
QQRRSST
5 7
AABBCCD
AFFBGGD
IIJBKKD
MNNOOPD
QQRRSST
6 7
ABCDEAB
ABCFEAB
AEAABAB
ACDAEEB
FFGAHIJ
KLMANOP
```

Sample Output

```
5
2
2
```

D

Problem B: Jack of All Trades

Jack Barter is a wheeler-dealer of the highest sort. He'll trade anything for anything, as long as he gets a good deal. Recently, he wanted to trade some red agate marbles for some goldfish. Jack's friend Amanda was willing to trade him 1 goldfish for 2 red agate marbles. But Jack did some more digging and found another friend Chuck who was willing to trade him 5 plastic shovels for 3 marbles while Amanda was willing to trade 1 goldfish for 3 plastic shovels. Jack realized that he could get a better deal going through Chuck (1.8 marbles per goldfish) than by trading his marbles directly to Amanda (2 marbles per goldfish).

Jack revels in transactions like these, but he limits the number of other people involved in a chain of transactions to 9 (otherwise things can get a bit out of hand). Normally Jack would use a little program he wrote to do all the necessary calculations to find the optimal deal, but he recently traded away his computer for a fine set of ivory-handled toothpicks. So Jack needs your help.

Input

Input will consist of multiple test cases. The first line of the file will contain an integer n indicating the number of test cases in the file. Each test case will start with a line containing two strings and a positive integer $m \leq 50$. The first string denotes the items that Jack wants, and the second string identifies the items Jack is willing to trade. After this will be m lines of the form

a_1 $name_1$ a_2 $name_2$

indicating that some friend of Jack's is willing to trade an amount a_1 of item $name_1$ for an amount a_2 of item $name_2$. (Note this does not imply the friend is also willing to trade a_2 of item $name_2$ for a_1 of item $name_1$.) The values of a_1 and a_2 will be positive and ≤ 20. No person will ever need more than $2^{31} - 1$ items to complete a successful trade.

Output

For each test case, output the phrase `Case i:` (where `i` is the case number starting at 1) followed by the best possible ratio that Jack can obtain. Output the ratio using 5 significant digits, rounded. Follow this by a single space and then the number of ways that Jack could obtain this ratio.

Sample Input

```
2
goldfish marbles 3
1 goldfish 2 marbles
5 shovels 3 marbles
1 goldfish 3 shovels
this that 4
7 this 2 that
14 this 4 that
7 this 2 theother
1 theother 1 that
```

Sample Output

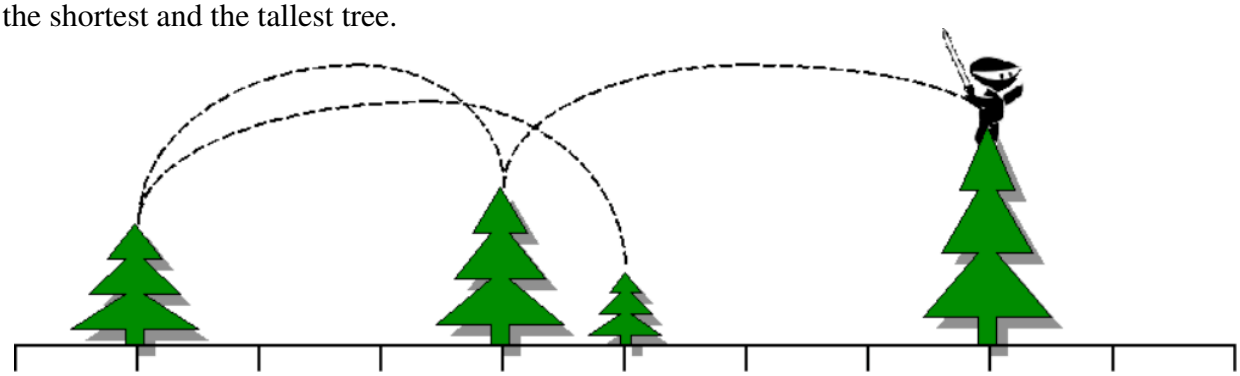
```
Case 1: 1.8000 1
Case 2: 0.28571 3
```

E

Problem D: The Ninja Way

As we all know, ninjas travel by jumping from treetop to treetop. A clan of ninjas plan to use N trees to hone their tree hopping skills. They will start at the shortest tree and make N-1 jumps, with each jump taking them to a taller tree than the one they're jumping from. When done, they will have been on every tree exactly once, and they will end up on the tallest tree.

The ninjas can travel for at most a certain horizontal distance D in a single jump. To make this as much fun as possible, the Ninjas want to maximize the distance between the planting positions of the shortest and the tallest tree.



The ninjas are going to plant the trees subject to the following constraints:

- All trees are to be planted along a one-dimensional path, which we can regard as the number line.
- Trees must be planted at integer locations along the path, with no two trees at the same location.
- Trees must be arranged so their planted ordering from left to right is the same as their ordering in the input: from $1, 2, \ldots, N$. They must **not** be sorted by height, or reordered in any way. They must be kept in their stated order.
- The Ninjas can only jump so far, so every tree must be planted close to the next tallest tree. In fact, they must be no further than D apart on the ground (the difference in their heights doesn't matter).

Given N trees, numbered $1, 2, \ldots, N$, each with a distinct integer height, help the ninjas figure out the maximum possible distance between the shortest tree and the tallest tree.

Input

Input will consist of multiple datasets. Each dataset begins with a line containing two integers N ($1 \leq N \leq 1000$) and D ($1 \leq D \leq 10^6$).

The next N lines each contains a single integer, giving the heights of the N trees.

The last test case is followed by a line with two zeros.

Output

For each test case, output a line with a single integer representing the maximum possible distance between the planted positions of the shortest and tallest tree, subject to the constraints above, or -1 if it is impossible to lay out the trees. Do not print any blank lines between answers.

Example

Given the input

```
4 4
20
30
10
40
5 6
20
34
54
10
15
4 2
10
20
16
13
0 0
```

the output would be

```
3
3
-1
```

G

Another Version of the Truth

Introduction:

Influence is a board game; while it can be played on almost any layout, an interesting layout is a hexagonal NxN grid, such that the layout resembles a rhombus.

An example 9x9 board is as follows:

```
1 2 3 4 5 6 7 8 9
 \ \ \ \ \ \ \ \ \
  * * * * * * * * * — A
   * * * * * * * * * — B
    * * * * * * * * * — C
     * * * * * * * * * — D
      * * * * * * * * * — E
       * * * * * * * * * — F
        * * * * * * * * * — G
         * * * * * * * * * — H
          * * * * * * * * * — I
```

On the above grid, F5 is adjacent to F4, F6, E5, E6, G4, and G5. (These coordinates are not used in the problem, but are useful for understanding the underlying adjacencies.)

The rules of *Influence* are simple; the pertinent details are as follows:

- Players take turns placing Manipulators on the board. Manipulators occupy a single location, and there may be at most one Manipulator at a location. If there is no empty location on the board to place a piece, the player must pass their turn.
- Each player has a certain amount of Influence. A player has a single point of Influence for every location on the board that is *strictly closer* to one of their Manipulators than a Manipulator of any other player. This is not "straight-line" distance, but the number of cells in a minimal path to a Manipulator. On the above grid, the cell F5 is 2 steps away from G6, 1 step away from G5, and 0 steps away from itself.
- The player with the most Influence at the end of the game wins.

In the sample input below, the first player (represented by "!" marks) has only two Influence, that provided by the locations that his Manipulators are on; the second player (represented by "@" signs) has ten Influence, and the third player (represented by "#" marks) has four Influence. There are nine locations that provide Influence to no player, as they are equally distant from two or more of the players.

Given a particular board layout, answer this question: what would the resulting Influence be for each player's optimal move if they were making the last move in the game?

Moves are to be considered independently; that is, the maximum score for the second player should be calculated based on the original board layout, *not* the one after the first player's best move.

Input:

Input to this problem will begin with a line containing a single integer N ($1 \le N \le 100$) indicating the number of data sets. Each data set consists of the following components:

- A line containing a single integer P ($2 \le P \le 4$) indicating the number of players in the game;
- A line containing a single integer D ($1 \le D \le 26$) indicating the board's dimension (9 would represent the 9x9 board above); and
- A series of D lines, each representing a row on the board from top to bottom. Each location on the row is represented by one of the following characters, separated by spaces:
  - `.` — An empty location;
  - `!` — A piece for the first player;
  - `@` — A piece for the second player;
  - `#` — A piece for the third player (if playing); or
  - `$` — A piece for the fourth player (if playing).

  Note that there may be extra whitespace on these lines (and only these lines). This is to make the input resemble the layout shown above.

Output:

For each data set in the input, output the heading "`DATA SET #`*K*" where K is 1 for the first data set, 2 for the second, etc. Then print P lines, each representing the maximum score possible for, in order, the first, second, third (if playing), and fourth (if playing) player if they were to make a single last move.

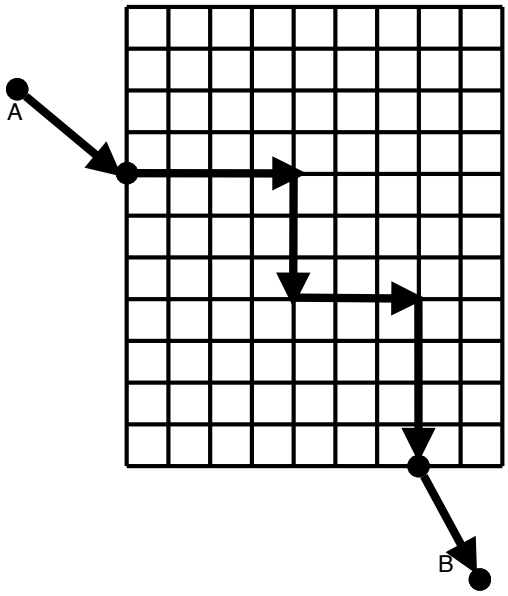
Sample Input:

```
1
3
5
! . . # .
 . @ . . !
  . . . . #
   . . @ . .
    . . . . .
```

Sample Output:

```
DATA SET #1
5
13
7
```

Problem E: Extended Manhattan Distance



The streets in Manhattan form a grid. If the grid is aligned such that the grid lines are parallel to the x- and y-coordinate axes, the distance one needs to walk or drive from one point to the other, assuming you can only move along streets and cannot take short cuts through buildings, equals $\Delta x + \Delta y$. This is called the *Manhattan distance*.

Now assume that the land outside the city grid is completely flat with no obstacles that prevent moving anywhere. Suppose we want to move from point A to point B where these points can be on the grid or outside the grid. When traveling outside the city, the shortest distance between the two points in this case will not necessarily be the Manhattan distance. It will be the Manhattan distance if the two points are both on the grid. If both points are, for example, north of the grid, the shortest distance between the two points will be the straight-line (Euclidean) distance between them. In other cases, calculating the distance may be more complicated.

In this problem, two opposite corners of the city grid will be specified. It will be assumed that the grid lines are parallel to the coordinate axes, and that the distance between any two consecutive grid lines, horizontal and vertical, is 1 unit. Two points A and B on the plane with integer coordinates will also be specified. Write a program to calculate the shortest distance between the two points, given that we can only move along the grid lines (i.e. in the city streets) within the city grid.

Input

Input will consist of multiple datasets. Each dataset will consist of a single line with eight integers, as follows:

$$x_L\ y_L\ x_U\ y_U\ x_A\ y_A\ x_B\ y_B$$

describing the points L, U, A, and B. L and U are the lower-left corner and the upper-right corner of the city grid, respectively. A and B are the two points between which we wish to travel.

All input integers will be in the range from -1000 to 1000 (inclusive), with $x_L < x_u$ and $y_L < y_U$. End of data will be signified by a line with eight zeros.

Output

For each data set, print one line containing the distance of the shortest path between the A and B, printed to to three decimal places of precision.

Example

Given the input

```
0 0 4 4 -1 0 5 3
0 0 4 4 2 2 5 3
0 0 0 0 0 0 0 0
```

the output would be

```
7.650
3.414
```

Problem G: The Worm Turns

Winston the Worm just woke up in a fresh rectangular patch of earth. The rectangular patch is divided into cells, and each cell contains either food or a rock. Winston wanders aimlessly for a while until he gets hungry; then he immediately eats the food in his cell, chooses one of the four directions (north, south, east, or west) and crawls in a straight line for as long as he can see food in the cell in front of him. If he sees a rock directly ahead of him, or sees a cell where he has already eaten the food, or sees an edge of the rectangular patch, he turns left or right and once again travels as far as he can in a straight line, eating food. He never revisits a cell. After some time he reaches a point where he can go no further so Winston stops, burps and takes a nap.

For instance, suppose Winston wakes up in the following patch of earth (X's represent stones, all other cells contain food):

	0	1	2	3	4
0					X
1					
2					
3		X	X		
4					

If Winston starts eating in row 0, column 3, he might pursue the following path (numbers represent order of visitation):

	0	1	2	3	4
0	4	3	2	1	X
1	5	18	17	16	15
2	6	19	20	21	14
3	7	X	X	22	13
4	8	9	10	11	12

In this case, he chose his path very wisely: every piece of food got eaten. Your task is to help Winston determine where he should begin eating so that his path will visit as many food cells as possible.

Input

Input will consist of multiple test cases. Each test case begins with two positive integers, m and n, defining the number of rows and columns of the patch of earth. Rows and columns are numbered starting at 0, as in the figures above. Following these is a non-negative integer r indicating the number of rocks, followed by a list of $2r$ integers denoting the row and column number of each rock. The last test case is followed by a pair of zeros. This should not be processed. The value $m \times n$ will not exceed 625.

Output

For each test case, print the test case number (beginning with 1), followed by four values:

```
amount row column direction
```

where `amount` is the maximum number of pieces of food that Winston is able to eat, `(row, column)` is the starting location of a path that enables Winston to consume this much food, and `direction` is

one of `E, N, S, W`, indicating the initial direction in which Winston starts to move along this path. If there is more than one starting location, choose the one that is lexicographically least in terms of row and column numbers. If there are optimal paths with the same starting location and different starting directions, choose the first valid one in the list `E, N, S, W`. Assume there is always at least one piece of food adjacent to Winston's initial position.

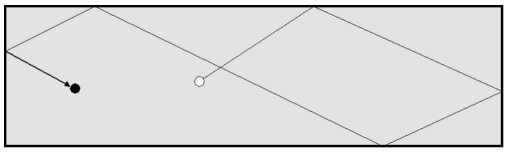
Sample Input

```
5 5
3
0 4 3 1 3 2
0 0
```

Sample Output

```
Case 1: 22 0 3 W
```

Problem F: Off the Wall



Consider a pool table, and the positions of a cue ball and a target ball. The cue ball must bounce off of a certain number of cushions (i.e. edges of the table), and then hit the target ball. What is the minimum distance that the cue ball has to travel?

Assume ideal cushions with perfect reflection (i.e., the angle at which a ball strikes the cushion is equal to the angle at which it bounces away), and a negligible ball diameter. The coordinate system uses a corner of the table as the origin, and the edges of the table are aligned with the coordinate axes. If the cue ball hits in a corner, it is considered to be hitting two cushions. The cue ball must hit exactly the right number of cushions first, before hitting the target ball.

Input

Input will consist of multiple datasets. Each dataset is on a single line containing seven integers:

$$L\ W\ x_C\ y_C\ x_T\ y_T\ N$$

The first two integers, L and W ($2 \leq L, W \leq 100$), are the dimensions of the table.

The next two pairs of integers are the x,y coordinates of the cue and target balls, respectively. You are guaranteed that $0 < x_C < L$, $0 < x_T < L$, $0 < y_C < W$, and $0 < y_T < W$. C and T are distinct points.

The final integer N, ($0 \leq N \leq 100$), is the number of cushions that must be hit prior to striking the target ball.

End of input will be indicated by a line with seven zeros.

Output

For each dataset, print a line with a single real number to 3 decimal digits precision, representing the shortest distance the cue ball must travel.

Example

Given the input

```
20 15 10 1 12 1 1
10 20 1 2 7 16 2
100 100 2 50 1 50 1
0 0 0 0 0 0 0
```

the output would be

```
2.828
19.698
100.005
```

J

Problem H: You'll be Working on the Railroad

Congratulations! Your county has just won a state grant to install a rail system between the two largest towns in the county — Acmar and Ibmar. This rail system will be installed in sections, each section connecting two different towns in the county, with the first section starting at Acmar and the last ending at Ibmar. The provisions of the grant specify that the state will pay for the two largest sections of the rail system, and the county will pay for the rest (if the rail system consists of only two sections, the state will pay for just the larger section; if the rail system consists of only one section, the state will pay nothing). The state is no fool and will only consider simple paths; that is, paths where you visit a town no more than once. It is your job, as a recently elected county manager, to determine how to build the rail system so that the county pays as little as possible. You have at your disposal estimates for the cost of connecting various pairs of cities in the county, but you're short one very important requirement — the brains to solve this problem. Fortunately, the lackeys in the computing services division will come up with something.

Input

Input will contain multiple test cases. Each case will start with a line containing a single positive integer $n \leq 50$, indicating the number of railway section estimates. (There may not be estimates for tracks between all pairs of towns.) Following this will be n lines each containing one estimate. Each estimate will consist of three integers s e c, where s and e are the starting and ending towns and c is the cost estimate between them. (Acmar will always be town 0 and Ibmar will always be town 1. The remaining towns will be numbered using consecutive numbers.) The costs will be symmetric, i.e., the cost to build a railway section from town s to town e is the same as the cost to go from town e to town s, and costs will always be positive and no greater than 1000. It will always be possible to somehow travel from Acmar to Ibmar by rail using these sections. A value of $n = 0$ will signal the end of input.

Output

For each test case, output a single line of the form

```
c1 c2 ... cm cost
```

where each `ci` is a city on the cheapest path and `cost` is the cost to the county (note `c1` will always be `0` and `cm` will always be `1` and `ci` and `ci+1` are connected on the path). In case of a tie, print the path with the shortest number of sections; if there is still a tie, pick the path that comes first lexicographically.

Sample Input

```
7
0 2 10
0 3 6
2 4 5
3 4 3
3 5 4
4 1 7
5 1 8
0
```

Sample Output

```
0 3 4 1 3
```

K